Experiment-3

AIM: Greedy Layerwise training of DNN

Code:

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, Input

from tensorflow.keras.utils import to\_categorical

import numpy as np

# 1. Data Preparation

# Load the dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Normalize the images to the range [0, 1]

x\_train = x\_train.astype('float32') / 255

x\_test = x\_test.astype('float32') / 255

# Flatten the images to 1D vectors of size 784 (28\*28)

x\_train = x\_train.reshape((x\_train.shape[0], 28 \* 28))

x\_test = x\_test.reshape((x\_test.shape[0], 28 \* 28))

# One-hot encode the labels

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

# 2. Layerwise Pre-training

def build\_autoencoder(input\_dim, encoding\_dim):

input\_img = Input(shape=(input\_dim,))

encoded = Dense(encoding\_dim, activation='relu')(input\_img)

decoded = Dense(input\_dim, activation='sigmoid')(encoded)

autoencoder = Model(input\_img, decoded)

encoder = Model(input\_img, encoded)

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

return autoencoder, encoder

# Layerwise training parameters

layer\_dims = [512, 256, 128]

input\_dim = x\_train.shape[1]

autoencoders = []

encoders = []

encoded\_input = x\_train

# Train each layer's autoencoder

for encoding\_dim in layer\_dims:

autoencoder, encoder = build\_autoencoder(input\_dim, encoding\_dim)

autoencoder.fit(encoded\_input, encoded\_input,

epochs=10,

batch\_size=256,

shuffle=True)

encoded\_input = encoder.predict(encoded\_input)

autoencoders.append(autoencoder)

encoders.append(encoder)

input\_dim = encoding\_dim

# 3. Model Definition and Initialization

# Define the DNN with pre-trained weights

model = Sequential()

# Add pre-trained layers

input\_dim = x\_train.shape[1]

for encoder in encoders:

model.add(Dense(encoder.layers[1].output\_shape[1], activation='relu', input\_shape=(input\_dim,)))

model.layers[-1].set\_weights(encoder.layers[1].get\_weights())

input\_dim = encoder.layers[1].output\_shape[1]

# Add the output layer

model.add(Dense(10, activation='softmax'))

# 4. Fine-tuning

# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_split=0.2)

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_accuracy}')

# Predict on the test set

predictions = model.predict(x\_test)

print(f'Predicted label: {np.argmax(predictions[0])}, True label: {np.argmax(y\_test[0])}')